CHAPTER 5

THE RELATIONAL DATABASE MODEL: INTRODUCTION

CHAPTER OBJECTIVES

After learning the material in this chapter, you will be able to:

- ✔ Explain why the relational database model became practical in about 1980.
- ✔ Define such basic relational database terms as relation and tuple.
- ✔ Describe the major types of keys including primary, candidate, and foreign.
- ✔ Describe how one-to-one, one-to-many, and many-to-many binary relationships are implemented in a relational database.
- ✔ Describe how relational data retrieval is accomplished in concept with the select, project, and join operators.
- ✔ Understand how the join operator facilitates data integration in relational database.

A Black& Decker power saw

BLACK & DECKER

Black & Decker is one of the world's largest producers of electric power tools and power tool accessories, is among the largest-selling residential lock manufacturers in the United States, and is a major manufacturer of faucets sold in the United States. It is also the world's largest producer of certain types of technology-based industrial fastening systems. The company's brand names include Black & Decker and DeWalt power tools,

Emhart Teknologies, Kwikset locks and other home security products, and Price Pfister plumbing fixtures. Based in Towson, MD, Black & Decker has manufacturing plants in ten countries and markets its products in over 100 countries around the globe.

One of the major factors in Black & Decker's Power Tools Division's leadership position is its highly advanced, database-focused information system that assures a steady and accurate supply of raw materials to the manufacturing floor. Using Manugistics' Demand and Supply Planning software, the system forecasts demand for Black & Decker's power tools and then generates a raw material supply plan based on the forecast and on the company's manufacturing capacity. These results are fed into SAP's Plant Planning System that takes into account suppliers' capabilities and lead-time constraints to set up orders for the raw materials.

Both the Manugistics and SAP software use Oracle databases to keep track of all of the data involved in these processes. Black & Decker runs the system, which became fully integrated in 1998, on clustered Compaq Alphas. The databases are also shared by the company's purchasing, receiving, finance, and accounting departments, assuring very high degrees of accuracy and speed throughout the company's operations and procedures. Included among the major database tables that support this information system are a material master table, a vendor master table, a bill-of-materials table (that indicates which parts go into making which other parts), a routing table (that indicates the work stations that the part will move through during manufacturing), planning, purchase order, customer, and other tables.

Printed by permission of Black & Decker

In 1970, Dr. Edgar F. (Ted) Codd of IBM published a paper entitled "A Relational Model of Data for Large Shared Data Banks" in *Communications of the ACM*. This paper marked the beginning of the field of relational database. During the 1970s, the relational approach to database progressed from being a technical curiosity to a subject of serious interest in the information systems community. But it was not until the early 1980s that commercially viable relational database management systems became available. There were two basic reasons for this lag. One was that while relational database was very tempting in concept, it was not easily applicable in a real-world environment for reasons related to performance, which we will discuss later in Chapter 8. The second reason was that at exactly the time that Codd's paper was published, the earlier hierarchical and network database management systems were just coming onto the commercial scene and were the focus of intense marketing efforts by the software and hardware vendors of the day.

Several factors converged in the early 1980s to begin turning the tide toward relational database. One was that the performance issues that held back its adoption in the 1970s began to be resolved. Another was that after a decade of use of hierarchical and network database management systems, information systems professionals were interested in finding an alternative that would help simplify the database design process and produce database structures that were easier to use and understand at all levels. At this time, too, there was increasing interest in having a DBMS environment that would allow easier, more intuitive access to the data by an increasingly broad range of personnel. Finally, the early 1980s saw the advent of the **personal computer (PC).** As software developers began trying to create all

manner of applications and supporting software utilities for the PC, it quickly became clear that the existing hierarchical and network database approaches would not work in the PC environment, for two reasons. One was that these DBMSs were simply too large to store and use on the early PCs. The other was that they were too complex to be used by the very broad array of noninformation systems professionals to whom the PCs were targeted.

Today, the relational approach to database management is by far the primary database management approach used in all levels of information systems and for most application purposes from accounting to banking to manufacturing to sales on the World Wide Web. Relational database management is represented today by such products as Microsoft Access and SQL Server, Oracle, Sybase, and IBM's DB2 and Informix. While these and other relational database systems have different features and implementations, they all share a common data structure philosophy and a common data access tool: Structured Query Language (SQL) (often pronounced "sequel"). This chapter will focus on the basic concepts of how data is stored and retrieved in a relational database by a relational DBMS. The following chapter, Chapter 6, will discuss some additional relational database concepts; Chapter 7 will describe logical database design; Chapter 8 will go into physical database design; and Chapter 9 will be devoted to SQL, primarily to its use in retrieving data from a relational database. *As explained in the Preface, it is important to note that Chapter 9 on SQL can be read and studied before the two design chapters, Chapters 7 and 8, without any loss of content or continuity.*

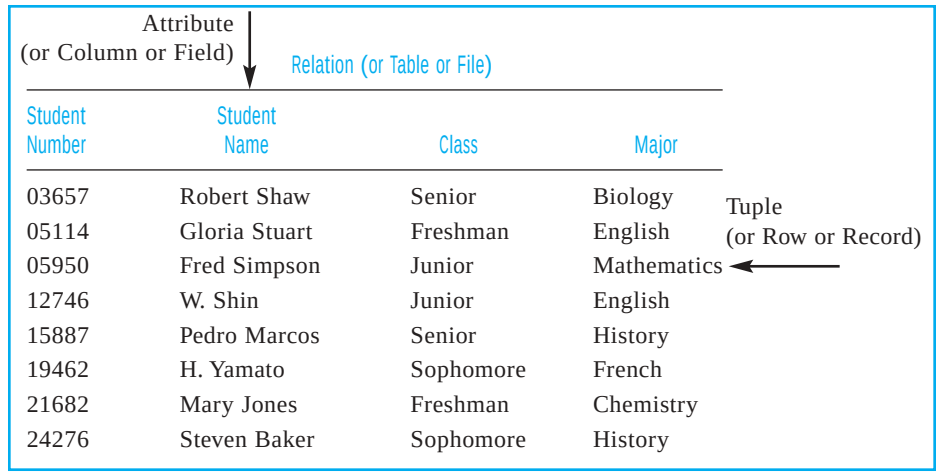
THE RELATIONAL DATABASE CONCEPT

Relational Terminology

In spite of the apparent conflict between non-redundant, linear file data storage and data integration demonstrated in the previous chapter, the concept of having the relative simplicity of simple, linear files or structures that resemble them in a true database environment is very desirable. After all, the linear file arrangement is the most basic and commonly used data structure available. This is precisely one of the advantages of relational database.

To begin with, consider the data structure used in relational databases. In a relational database, the data *appears* to be stored in what we have been referring to as simple, linear files. Following the conventions of the area of mathematics on which relational database is based, we will begin calling those simple linear files **relations,** although in common practice they are also referred to as tables. In the terminology of files, each **row** is called a record, while in a relation, each row is called a **tuple.** In files, each **column** is called a field, while in a relation each column is called an **attribute.** In practice, in speaking about relational database, it is common for people to use relation, table, and file synonymously. Similarly, tuple, row, and record are often used synonymously, as are attribute, column, and field, (Figure 5.1). We will use an appropriate term in each particular situation during our discussion. In particular, we will use the term relation in this chapter and the next, in which we are talking about relational database concepts. Following common usage, we will generally use the term table in the more applied parts of the book, such as in the corporate database stories in each chapter and in the discussion of SQL in Chapter 9.

➤ **Figure 5.1**
Relational database terminology



Technical differences exist between the concept of a file and the concept of a relation (which is why we say that in a relational database the data only *appears* to be stored in structures that look like files). The differences include:

- The columns of a relation can be arranged in any order without affecting the meaning of the data. This is not true of a file.
- Similarly, the rows of a relation can be arranged in any order, which is not true of a file.
- Every row/column position, sometimes referred to as a cell, can have only a single value, which is not necessarily true in a file.
- No two rows of a relation are identical, which is not necessarily true in a file.

A relational database is simply a collection of relations that, as a group, contain the data that describes a particular business environment.

Primary and Candidate Keys

Primary Keys Figure 5.2 contains two relations, the SALESPERSON relation and the CUSTOMER relation, from General Hardware Company's relational database. The SALESPERSON relation has four rows, each representing one salesperson. Also, the SALESPERSON relation has four columns, each representing a characteristic of salespersons. Similarly, the CUSTOMER relation has nine rows, each representing a customer, and four columns.

A relation always has a unique **primary key.** A primary key (sometimes shortened in practice to just "the key") is an attribute or a group of attributes whose values are unique throughout all of the rows of the relation. The primary key represents the characteristic of a collection of entities that uniquely identifies each one. For example, in the situation described by the relations of Figure 5.2, each salesperson has been assigned a unique salesperson number and each customer has been assigned a unique customer number. Therefore the Salesperson Number attribute is the primary key of the SALESPERSON relation, and the Customer Number attribute is the primary key of the CUSTOMER relation. As shown in Figure 5.2, we will start marking the primary key attribute(s) with a single, solid underline.

➤ **Figure 5.2**
General Hardware Company relational database

(a) SALESPERSON relation

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire
137	Baker	10	1995
186	Adams	15	2001
204	Dickens	10	1998
361	Carlyle	20	2001

(b) CUSTOMER relation

Customer Number	Customer Name	Salesperson Number	HQ City
0121	Main St. Hardware	137	New York
0839	Jane's Stores	186	Chicago
0933	ABC Home Stores	137	Los Angeles
1047	Acme Hardware Store	137	Los Angeles
1525	Fred's Tool Stores	361	Atlanta
1700	XYZ Stores	361	Washington
1826	City Hardware	137	New York
2198	Western Hardware	204	New York
2267	Central Stores	186	New York

The number of attributes involved in the primary key is always the minimum number of attributes that provide the uniqueness quality. For example, in the SALESPERSON relation, it would make no sense to have the combination of Salesperson Number and Salesperson Name as the primary key because Salesperson Number is unique by itself. However, consider the situation of a SALESPERSON relation that does not include a Salesperson Number attribute, but instead has a First Name attribute, a Middle Name attribute, and a Last Name attribute. The primary key might then be the combination of the First, Middle, and Last Name attributes. (Assuming this would always produce a unique combination of values. If it did not, then a fourth attribute could be added to the relation and to the primary key as a sequence field to produce, for example, John Alan Smith #1, John Alan Smith #2, and so forth.) Some attribute or combination of attributes of a relation has to be unique and can serve as the unique primary key, since, by definition, no two rows can be identical. In the worst case, all of the relation's attributes combined could serve as the primary key if necessary (but this situation is uncommon in practice).

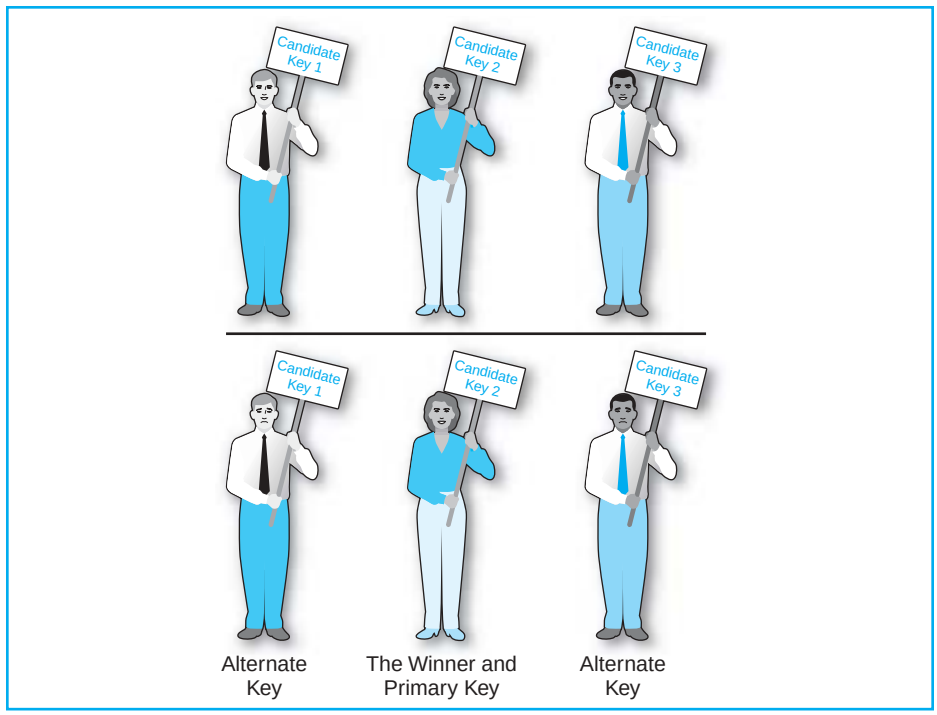
Candidate Keys If a relation has more than one attribute or minimum group of attributes that represents a way of uniquely identifying the entities, then they are each called a *candidate key.* (Actually, if there is only one unique attribute or minimum group of attributes, it can also be called a candidate key.) For example, in a personnel relation, an Employee Number attribute and a Social Security number attribute (each of which is obviously unique) would each be a candidate key of that

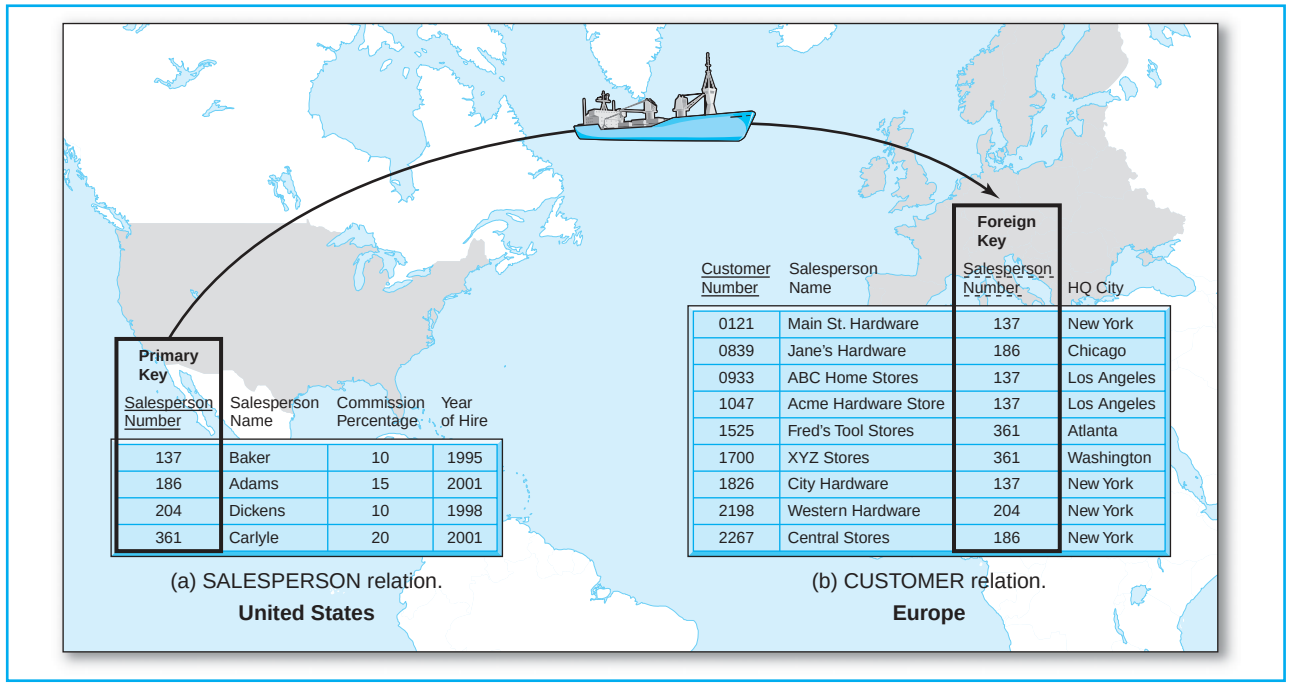
relation. When there is more than one candidate key, one of them must be chosen to be the primary key of the relation. That is where the term candidate key comes from since each one is a candidate for selection as the primary key. The decision of which candidate key to pick to be the primary key is typically based on which one will be the best for the purposes of the applications that will use the relation and the database. Sometimes the term **alternate key** is used to describe a candidate key that was not chosen to be the primary key of the relation (Figure 5.3).

Foreign Keys and Binary Relationships

Foreign Keys If in a collection of relations that make up a relational database, an attribute or group of attributes serves as the primary key of one relation and also appears in another relation, then it is called a *foreign key* in that other relation. Thus Salesperson Number, which is the primary key of the SALESPERSON relation, is considered a foreign key in the CUSTOMER relation (Figure 5.4). As shown in Figure 5.4, we will start marking the Foreign Key attribute(s) with a dashed underline. The concept of the foreign key is crucial in relational database, because the foreign key is the mechanism that ties relations together to represent unary, binary, and ternary relationships. We begin the discussion by considering how binary relationships are stored in relational databases. These are both the most common and the easiest to deal with. The unary and ternary relationships will come later. Recall from the discussion of the entity-relationship model that the three kinds of binary relationships among the entities in the business environment are the one-to-one, one-to-many, and many-to-many relationships. The first case is the one-to-many relationship, which is typically the most common of the three.

➤ **Figure 5.3**
Candidate keys become either primary or alternate keys





➤ **Figure 5.4** A foreign key

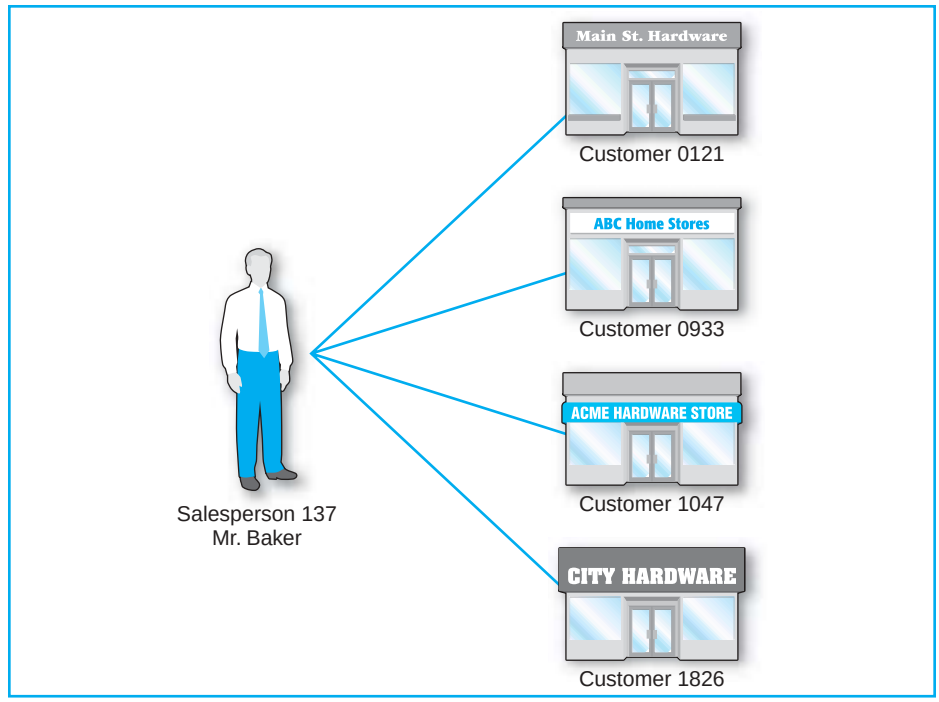
One-to-Many Binary Relationship Consider the SALESPERSON and CUSTOMER relations of Figure 5.2, repeated in Figure 5.4. As one would expect in most sales-oriented companies, notice that each salesperson is responsible for several customers, while each customer has a single salesperson as their point of contact with General Hardware. This one-to-many binary relationship can be represented as:

Salesperson ⟷ Customer

For example, the Salesperson Number attribute of the CUSTOMER relation shows that salesperson 137 is responsible for customers 0121, 0933, 1047, and 1826. Looking at it from the point of view of the customer, the same relation shows that the only salesperson associated with customer 0121 is salesperson 137 (Figure 5.5). This last point has to be true. After all, there is only one record for each customer in the CUSTOMER relation (the Customer Number attribute is unique since it is the relation's primary key), and there is only one place to put a salesperson number in it. The bottom line is that the Salesperson Number foreign key in the CUSTOMER relation effectively establishes the one-to-many relationship between salespersons and customers.

Notice that, in this case, the primary key of the SALESPERSON relation and the corresponding foreign key in the CUSTOMER relation both have the same **attribute name,** Salesperson Number. This will often be the case, but it does not have to be. What is necessary is that both attributes have the same **domain of values;** that is they must both have values of the same type, such as, in this case, three-digit whole numbers that are the identifiers for salespersons.

➤ **Figure 5.5**
A salesperson and his four customers



It is the presence of a salesperson number in a customer record that indicates which salesperson the customer is associated with. Fundamentally, that is why the Salesperson Number attribute is in the CUSTOMER relation, and that is the essence of its being a foreign key in that relation. Later in the book, we will discuss database design issues in detail. But for now, note that when building a one-to-many relationship into a relational database, it will always be the case that the unique identifier of the entity on the "one side" of the relationship (Salesperson Number, in this example) will be placed as a foreign key in the relation representing the entity on the "many side" of the relationship (the CUSTOMER relation, in this example).

Here's something else about foreign keys. In some situations a relation doesn't have a single, unique attribute to serve as its primary key. Then, it requires a combination of two or more attributes to reach uniqueness and serve as its primary key. Sometimes one or more of the attributes in that combination can be a foreign key! Yes, when this happens, a foreign key is actually part of the relation's primary key! This was not the case in the CUSTOMER relation of Figure 5.2b. In this relation, the primary key only consists of one attribute, Customer Number, which is unique all by itself. The foreign key, Salesperson Number, is clearly not a part of the primary key.

Here is an example of a situation in which a foreign key is part of a relation's primary key. Figure 5.6 adds the CUSTOMER EMPLOYEE relation (Figure 5.6c), to the General Hardware database. Remember that General Hardware's customers are the hardware stores, home improvement stores, or chains of such stores that it supplies. Figure 5.6c, the CUSTOMER EMPLOYEE relation, lists the employees of each of General Hardware's customers. In fact, there is a one-to-many relationship between customers and customer employees. A customer (like a hardware store) has many employees, but an employee, a person, works in only one store:

Customer ⟷ **Customer Employee**

➤ **Figure 5.6**
General Hardware Company relational database including the CUSTOMER EMPLOYEE relation

(a) SALESPERSON relation

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire
137	Baker	10	1995
186	Adams	15	2001
204	Dickens	10	1998
361	Carlyle	20	2001

(b) CUSTOMER relation

Customer Number	Customer Name	Salesperson Number	HQ City
0121	Main St. Hardware	137	New York
0839	Jane's Stores	186	Chicago
0933	ABC Home Stores	137	Los Angeles
1047	Acme Hardware Store	137	Los Angeles
1525	Fred's Tool Stores	361	Atlanta
1700	XYZ Stores	361	Washington
1826	City Hardware	137	New York
2198	Western Hardware	204	New York
2267	Central Stores	186	New York

(c) CUSTOMER EMPLOYEE relation

Customer Number	Employee Number	Employee Name	Title
0121	27498	Smith	Co-Owner
0121	30441	Garcia	Co-Owner
0933	25270	Chen	VP Sales
0933	30441	Levy	Sales Manager
0933	48285	Morton	President
1525	33779	Baker	Sales Manager
2198	27470	Smith	President
2198	30441	Jones	VP Sales
2198	33779	Garcia	VP Personnel
2198	35268	Kaplan	Senior Accountant

For example, Figure 5.6c shows that customer 2198 has four employees—Smith, Jones, Garcia, and Kaplan. Each of those people is assumed to work for only one customer company, customer 2198. Following the rule we developed for setting up a one-to-many relationship with a foreign key, the Customer attribute must appear in the CUSTOMER EMPLOYEE relation as a foreign key, and indeed it does.

Now, what about finding a legitimate primary key for the CUSTOMER EMPLOYEE relation? The assumption here is that employee numbers *are only unique within a company*; they are not unique across all of the customer companies.

Thus, as shown in the CUSTOMER EMPLOYEE relation of Figure 5.6c, there can be an employee of customer number 0121 who is employee number 30441 in that company's employee numbering system, an employee of customer number 0933 who is employee number 30441 in that company's system, and also an employee of customer number 2198 who is also employee number 30441. That being the case, the Employee Number is not a **unique attribute** in this relation. Neither it nor any other single attribute of the CUSTOMER EMPLOYEE relation is unique and can serve, alone, as the relation's primary key. But the combination of Customer Number and Employee Number is unique. After all, we know that customer numbers are unique and that within each customer company, employee numbers are unique. That means that, as shown in Figure 5.6c, the combination of Customer Number and Employee Number can be and is the relation's primary key. That also means that Customer Number is both a foreign key in the CUSTOMER EMPLOYEE relation *and* a part of its primary key. As shown in Figure 5.6c, we will start marking attributes that are both a foreign key and a part of the primary key with an underline consisting of a dashed line over a solid line.

Many-to-Many Binary Relationship

Storing the Many-to-Many Binary Relationship Figure 5.7 expands the General Hardware database by adding two more relations: the PRODUCT relation (Figure 5.7d), and the SALES relation (Figure 5.7e). The PRODUCT relation simply lists the products that General Hardware sells, one row per product, with Product Number as the unique identifier and thus the primary key of the relation. Each of General Hardware's salespersons can sell any or all of the company's products, and each product can be sold by any or all of its salespersons. Therefore the relationship between salespersons and products is a many-to-many relationship.

Salesperson ⟷ **Product**

So, the database will somehow have to keep track of this many-to-many relationship between salespersons and products. The way that a many-to-many relationship is represented in a relational database is by the creation of an additional relation, in this example, the SALES relation in Figure 5.7e. The SALES relation of Figure 5.7e is intended to record the *lifetime* sales of a particular product by a particular salesperson. Thus there will be a single row in the relation for each applicable combination of salesperson and product (i.e., when a particular salesperson *has actually sold* some of the particular product). For example, the first row of the SALES relation indicates that salesperson 137 has sold product 19440. Since it is sufficient to record that fact once, the combination of the Salesperson Number and Product Number attributes always produces unique values. So, in this case, the new relation created to record the many-to-many relationship will have as its primary key the combined, unique identifiers of the two entities in the many-to-many relationship. That's why, in this example, the Salesperson Number and Product Number attributes both appear in the SALES relation. Each of the two is a foreign key in the SALES relation since each is the primary key of another relation in the database. The combination of these two attributes is unique and, combined, they comprise the primary key of the newly created SALES relation.

Figure 5.7
General Hardware Company relational database including the PRODUCT and SALES relation

(a) SALESPERSON relation

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire
137	Baker	10	1995
186	Adams	15	2001
204	Dickens	10	1998
361	Carlyle	20	2001

(b) CUSTOMER relation

Customer Number	Customer Name	Salesperson Number	HQ City
0121	Main St. Hardware	137	New York
0839	Jane's Stores	186	Chicago
0933	ABC Home Stores	137	Los Angeles
1047	Acme Hardware Store	137	Los Angeles
1525	Fred's Tool Stores	361	Atlanta
1700	XYZ Stores	361	Washington
1826	City Hardware	137	New York
2198	Western Hardware	204	New York
2267	Central Stores	186	New York

(c) CUSTOMER EMPLOYEE relation

Customer Number	Employee Number	Employee Name	Title
0121	27498	Smith	Co-Owner
0121	30441	Garcia	Co-Owner
0933	25270	Chen	VP Sales
0933	30441	Levy	Sales Manager
0933	48285	Morton	President
1525	33779	Baker	Sales Manager
2198	27470	Smith	President
2198	30441	Jones	VP Sales
2198	33779	Garcia	VP Personnel
2198	35268	Kaplan	Senior Accountant

(Continues)

The new SALES relation of Figure 5.7e effectively records the many-to-many relationship between salespersons and products. This is illustrated from the "salesperson side" of the many-to-many relationship by looking at the first three rows of the SALES relation and seeing that salesperson 137 sells products 19440, 24013, and 26722. It is illustrated from the "product side" of the many-to-many relationship by scanning down the Product Number column of the SALES relation, looking for the value 19440, and seeing that product 19440 is sold by salespersons 137 and 186 (Figure 5.8).

➤ **Figure 5.7 (Continued)** General Hardware Company relational database including the PRODUCT and SALES relation

(d) PRODUCT relation

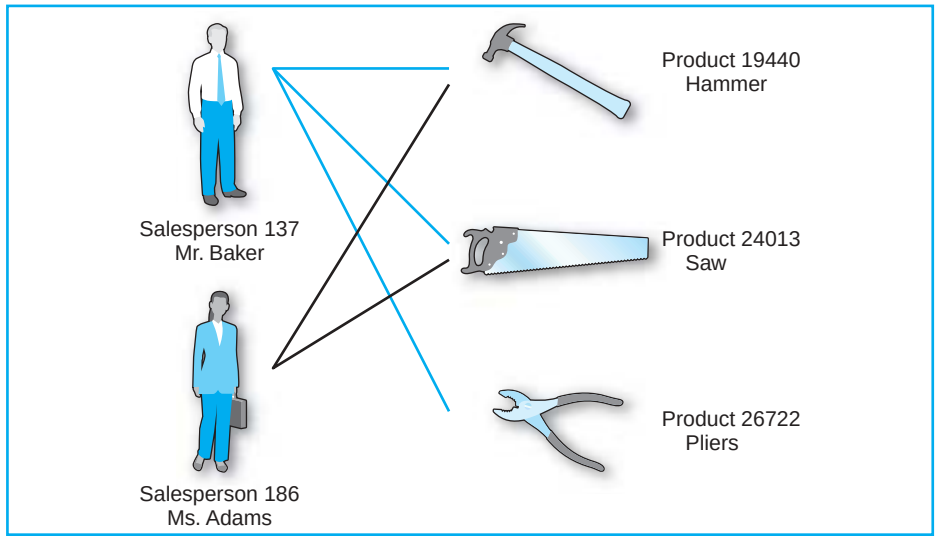
Product Number	Product Name	Unit Price
16386	Wrench	12.95
19440	Hammer	17.50
21765	Drill	32.99
24013	Saw	26.25
26722	Pliers	11.50

(e) SALES relation

Salesperson Number	Product Number	Quantity
137	19440	473
137	24013	170
137	26722	688
186	16386	1,745
186	19440	2,529
186	21765	1,962
186	24013	3,071
204	21765	809
204	26722	734
361	16386	3,729
361	21765	3,110
361	26722	2,738

Intersection Data What about the Quantity attribute in the SALES relation? In addition to keeping track of which salespersons have sold which products, General Hardware wants to record *how many* of each particular product each salesperson has sold since the product was introduced or since the salesperson joined the company. So, it sounds like there has to be a Quantity attribute. And, an attribute describes an entity, right? Then, which entity does the Quantity attribute describe? Does it describe salespersons the way the Year of Hire does in the SALESPERSON relation? Does it describe products the way Unit Price does in the PRODUCT relation? Each salesperson has exactly one date of hire, and each product has exactly one unit price. But a salesperson doesn't have just one quantity associated with her because she sells many products, and similarly, a product doesn't have just one quantity associated with it because it is sold by many salespersons.

➤ **Figure 5.8** Many-to-many relationship between salespersons and products as shown in the SALES relation

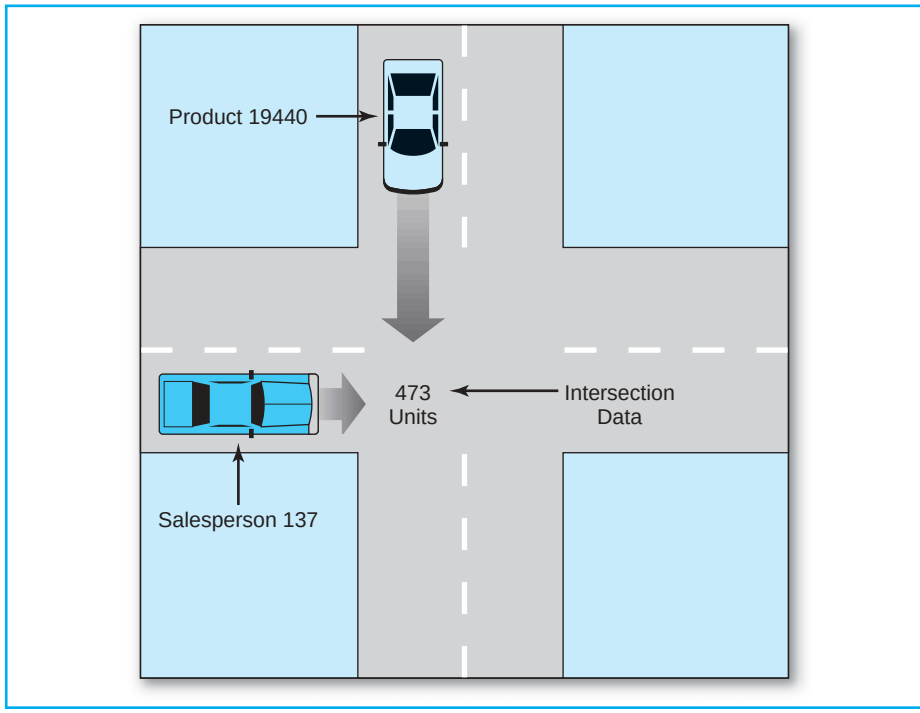


While year of hire is clearly a characteristic of salespersons and unit price is clearly a characteristic of products, quantity is a characteristic of *the relationship between salesperson and product*. For example, the fact that salesperson 137 appears in the first row of the SALES relation of Figure 5.7e along with product 19440 indicates that he has a history of selling this product. But do we know more about his history of selling it? Yes! That first row of Figure 5.7e indicates that salesperson 137 has sold 473 units of product 19440. Quantity *describes the many-to-many relationship* between salespersons and products. In a sense, it falls at the intersection between the two entities and is called intersection data (Figure 5.9).

Since the many-to-many relationship has its own relation in the database and since it can have attributes, does that mean that we should think of it as a kind of entity? Yes! Many people do just that and refer to it as an associative entity, a concept that we first described when discussing data modeling in Chapter 3.

Additional Many-to-Many Concepts Before leaving the subject of many-to-many relationships, there are a few more important points to make. First, will the combination of the two primary keys representing the two entities in the many-to-many relationship always serve as a unique identifier or primary key in the additional relation representing the many-to-many relationship? That depends on the precise nature of the many-to-many relationship. For example, in the situation of the SALES relation in Figure 5.7e, the combination of the two entity identifier attributes works perfectly as the primary key, as described above. But what if General Hardware decides that it wants to keep track of each salesperson's *annual* sales of each product instead of their *lifetime* sales? Fairly obviously, a new attribute, Year, would have to

Figure 5.9
Intersection data that indicates that salesperson 137 has sold 473 units of product 19440



➤ **Figure 5.10**
Modified SALES relation of the General Hardware Company relational database, including a Year attribute

SALES relation (modified)

Salesperson Number	Product Number	Year	Quantity
137	19440	1999	132
137	19440	2000	168
137	19440	2001	173
137	24013	2000	52
137	24013	2001	118
137	26722	1999	140
137	26722	2000	203
137	26722	2001	345
186	16386	1998	250
186	16386	1999	245
186	16386	2000	581
186	16386	2001	669

be added to the SALES relation, as shown in Figure 5.10. Moreover, as demonstrated by a few sample rows of that relation, the combination of Salesperson Number and Product Number is no longer unique. For example, salesperson 137 sold many units of product 19440 in each of 1999, 2000, and 2001. The first three records of the relation all have the salesperson number, product number combination of 137, 19440. The way to solve the problem in this instance is to add the Year attribute to the Salesperson Number and Product Number attributes to form a three-attribute, unique, primary key. It is quite common in practice to have to add such a "timestamp" to a relation storing a many-to-many relationship in order to attain uniqueness and have a legitimate primary key. Sometimes, as in the example in Figure 5.10, this is accomplished with a Year attribute. A Date attribute is required if the data may be stored two or more times in a year. A Time attribute is required if the data may be stored more than once in a day.

Next is the question of why an additional relation is necessary to represent a many-to-many relationship. For example, could the many-to-many relationship between salespersons and products be represented in either the SALESPERSON or PRODUCT relation? The answer is no! If, for instance, you tried to represent the many-to-many relationship in the SALESPERSON relation, you would have to list all of the products (by Product Number) that a particular salesperson has sold in that salesperson's record. Furthermore, in some way you would have to carry the Quantity intersection data along with it. For example, in the SALESPERSON relation, the row for salesperson 137 would have to be extended to include products 19440, 24013, and 26722, plus the associated intersection data (Figure 5.11a). Alternately, one could envision a single additional attribute in the SALESPERSON relation into which all of the related product number and intersection data for each salesperson would somehow be stuffed (Figure 5.11b), although, aside from other problems, this would violate the rule about every cell in a relation having only a single value. In either case, it would be unworkable. Because, in general, each salesperson has been involved in selling different numbers of product types, each record of the SALESPERSON relation would be a different length. Furthermore, additions, deletions, and updates of product/quantity pairs would be a nightmare. Also, trying to access the related data from the "product side," for example, looking for all of the salespersons who have sold a particular product, would be very difficult. And, incidentally, trying to make this work by putting the salesperson data into the PRODUCT relation, instead of putting the product data into the SALESPERSON relation as in Figure 5.11, would generate an identical set of problems. No, the only way that's workable is to

(a) Additional Product and Quantity columns

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Product	Qty	Product	Qty	Product	Qty	Product	Qty
137	Baker	10	1995	19440	473	24013	170	26722	688		
186	Adams	15	2001	16386	1745	19440	2529	21765	1962	24013	3071
204	Dickens	10	1998	21765	809	26722	734				
361	Carlyle	20	2001	16386	3729	21765	3110	26722	2738		

(b) One additional column for Product and Quantity Pairs

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Product and Quantity Pairs
137	Baker	10	1995	(19440, 473) (24013, 170) (26722, 688)
186	Adams	15	2001	(16386, 1745) (19440, 2529) (21765, 1962) (24013, 3071)
204	Dickens	10	1998	(21765, 809) (26722, 734)
361	Carlyle	20	2001	(16386, 3729) (21765, 3110) (26722, 2738)

➤ **Figure 5.11** Unacceptable ways of storing a binary many-to-many relationship

create an additional relation to represent the many-to-many relationship. Each combination of a related salesperson and product has its own record, making the insertion, deletion, and update of related items feasible, providing a clear location for intersection data, and avoiding the issue of variable-length records.

Finally, there is the question of whether an additional relation is required to represent a many-to-many relationship in the case where there is no intersection data. For example, suppose that General Hardware wants to track which salespersons have sold which products but has no interest in how many units of each product they have sold. The SALES relation of Figure 5.7e would then have only the Salesperson Number and Product Number attributes (Figure 5.12). Could this information be stored in some way other than with the additional SALES relation? The answer is that the additional relation is still required. Note that in the preceding explanation, of why an additional relation is necessary in general to represent a many-to-many relationship, the intersection data played only a small role. The issues would still be there, even without intersection data.

One-to-One Binary Relationship After considering one-to-many and many-to-many binary relationships in relational databases, the remaining binary relationship is the one-to-one relationship. Each of General Hardware's salespersons has exactly one office, and each office is occupied by exactly one salesperson (Figure 5.13).

Salesperson ⟷ Office

Figure 5.14 shows the addition of the OFFICE relation (Figure 5.14f) to the General Hardware relational database. The SALESPERSON relation has the Office Number attribute as a foreign key so that the company can look up the record for a salesperson and see to which office she is assigned. Because this is a one-to-one relationship and each salesperson has only one office, the company can also scan down

➤ **Figure 5.12**
The many-to-many SALES relation without intersection data

SALES relation (without intersection data)

Salesperson Number	Product Number
137	19440
137	24013
137	26722
186	16386
186	19440
186	21765
186	24013
204	21765
204	26722
361	16386
361	21765
361	26722

the Office Number column of the SALESPERSON relation, find a particular office number (which can only appear once, since it's a one-to-one relationship), and see which salesperson is assigned to that office. In general, this is the way that one-to-one binary relationships are built into relational databases. The unique identifier, the primary key, of one of the two entities in the one-to-one relationship is inserted into the other entity's relation as a foreign key. The question of which of the two entities is chosen as the "donor" of its primary key and which is chosen as the "recipient" will be discussed further when we talk about logical design in Chapter 7.

But there is another interesting question about this arrangement. Could the SALESPERSON and OFFICE relations of Figure 5.14 be combined into one relation? After all, a salesperson has only one office and an office has only one salesperson assigned to it. So, if an office and its unique identifier, Office Number, "belongs" to

➤ **Figure 5.13**
A one-to-one binary relationship

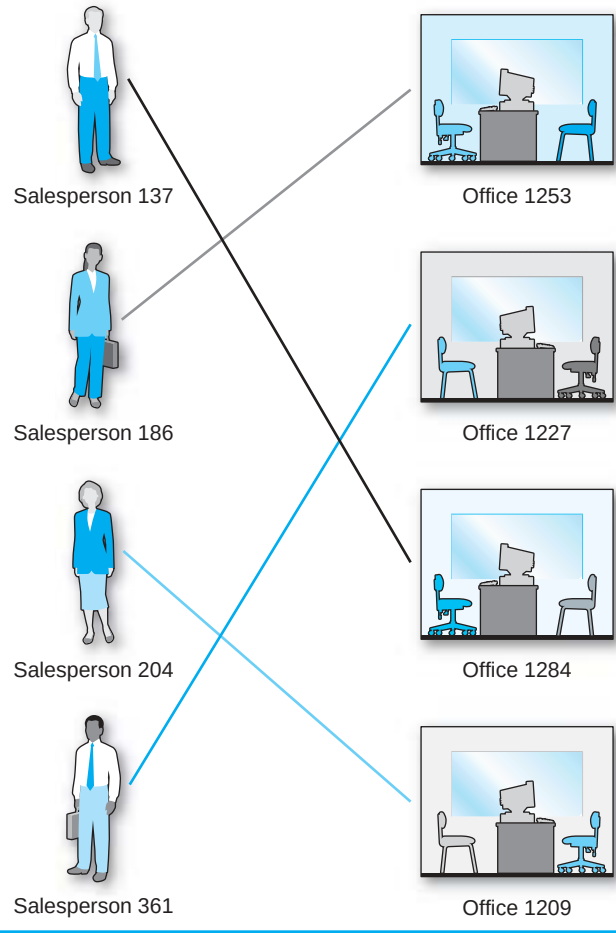


Figure 5.14
General Hardware Company relational database including the OFFICE relation

(a) SALESPERSON relation

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Office Number
137	Baker	10	1995	1284
186	Adams	15	2001	1253
204	Dickens	10	1998	1209
361	Carlyle	20	2001	1227

(b) CUSTOMER relation

Customer Number	Customer Name	Salesperson Number	HQ City
0121	Main St. Hardware	137	New York
0839	Jane's Stores	186	Chicago
0933	ABC Home Stores	137	Los Angeles
1047	Acme Hardware Store	137	Los Angeles
1525	Fred's Tool Stores	361	Atlanta
1700	XYZ Stores	361	Washington
1826	City Hardware	137	New York
2198	Western Hardware	204	New York
2267	Central Stores	186	New York

(c) CUSTOMER EMPLOYEE relation

Customer Number	Employee Number	Employee Name	Title
0121	27498	Smith	Co-Owner
0121	30441	Garcia	Co-Owner
0933	25270	Chen	VP Sales
0933	30441	Levy	Sales Manager
0933	48285	Morton	President
1525	33779	Baker	Sales Manager
2198	27470	Smith	President
2198	30441	Jones	VP Sales
2198	33779	Garcia	VP Personnel
2198	35268	Kaplan	Senior Accountant

(Continues)

one particular salesperson, so does that office's Telephone Number and Size. Indeed, when we want to be able to contact a salesperson, we ask for *her phone number*, not for "her office's phone number!" So, could we combine the SALESPERSON and OFFICE relations of Figure 5.14 into the single relation of Figure 5.15? The answer is, it's possible in some cases, but you have to be very careful about making such a decision. In the General Hardware case, how would you store an *unoccupied* office in the database? The relation of Figure 5.15 only

➤ **Figure 5.14 (Continued)** General Hardware Company relational database including the OFFICE relation

(d) PRODUCT relation

Product Number	Product Name	Unit Price
16386	Wrench	12.95
19440	Hammer	17.50
21765	Drill	32.99
24013	Saw	26.25
26722	Pliers	11.50

(e) SALES relation

Salesperson Number	Product Number	Quantity
137	19440	473
137	24013	170
137	26722	688
186	16386	1,745
186	19440	2,529
186	21765	1,962
186	24013	3,071
204	21765	809
204	26722	734
361	16386	3,729
361	21765	3,110
361	26722	2,738

(f) OFFICE relation

Office Number	Telephone	Size (sq. ft.)
1253	901-555-4276	120
1227	901-555-0364	100
1284	901-555-7335	120
1209	901-555-3108	95

allows data about an office to be stored if the office is *occupied*. After all, the primary key of Figure 5.15's relation is Salesperson Number! You can't have a record with office data in it and no salesperson data. A case where it might work is a database of U.S. states and their governors. Every state *always* has exactly one governor, and anyone who is a governor *must* be associated with one state. There can't be a state without a governor or a governor without a state.

At any rate, in practice, there are a variety of reasons for keeping the two relations involved in the one-to-one relationship separate. It may be that because each of the two entities involved is considered sufficiently important in its own right, it simply adds clarity to the database. It may be because most users at any one time seek data about only one of the two entities. It may have to do with splitting the data between different geographic sites. It can even be done for system performance purposes in the case where the records would be unacceptably long if the data was all contained in one relation. These issues will be discussed later in this book, but at this point it is important to have at least a basic idea of the intricacies of the one-to-one relationship.

DATA RETRIEVAL FROM A RELATIONAL DATABASE

Extracting Data from a Relation

Thus far, the discussion has concentrated on how a relational database is structured, but building relations and loading them with data is only half of the story. The other half is the effort to retrieve the data in a way that is helpful and beneficial to the business organization that built the database. If the database management system did not provide any particular help with this effort, then the problem would revert to simply writing a program in some programming language to retrieve data from the relations, treating them as if they were simple, linear files. But the crucial point

➤ **Figure 5.15**
Combining the SALESPERSON and OFFICE relations into a single relation

Combined SALESPERSON/OFFICE relation

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Office Number	Telephone	Size (sq. ft.)
137	Baker	10	1995	1284	901-555-7335	120
186	Adams	15	2001	1253	901-555-4276	120
204	Dickens	10	1998	1209	901-555-3108	95
361	Carlyle	20	2001	1227	901-555-0364	100

is that a major, defining feature of a relational DBMS is the ability to accept high-level **data retrieval** commands, process them against the database's relations, and return the desired data. The data retrieval mechanism is a built-in part of the DBMS and does not have to be written from scratch by every program that uses the database. As we shall soon see, this is true even to the extent of matching related records in different relations (integrating data), as in the earlier example of finding the *name* of the salesperson on a particular customer account. We shall address what relational retrieval might look like, first in terms of single relations and then across multiple relations.

Since a relation can be viewed as a tabular or rectangular arrangement of data values, it would seem to make sense to want to approach data retrieval horizontally, vertically, or in a combination of the two. To take a horizontal slice of a relation implies retrieving one or more rows of the relation. In effect, that's an expression for retrieving one or more records or retrieving the data about one or more entities. Taking a vertical slice of a relation means retrieving one or more entire columns of the relation (down through all of its rows). Taken in combination, we can retrieve one or more columns of one or more rows, the minimum of which is a single column of a single row, or a single attribute value of a single record. That's as fine a sense of retrieval, as we would ever want.

Using terminology from a database formalism called **relational algebra,** and an informal, hypothetical command style for now, there are two commands called *Select* and *Project,* which are capable of the kinds of horizontal and vertical manipulations that were just suggested. (*Note:* The use of the word "Select" here is *not* the same as its use in the SQL data retrieval language, which we will discuss later in this book.)

The Relational Select Operator

Consider the database of Figure 5.14 and its SALESPERSON relation (Figure 5.14a). To begin with, suppose that we want to find the row or record for salesperson number 204. In a very straightforward way, the informal command might be:

> Select rows from the SALESPERSON relation in which Salesperson Number = 204.

The result would be:

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire
204	Dickens	10	1998

Notice that the result of the Select operation is itself a relation, in this case consisting of only one row. *The result of a relational operation will always be a relation,* whether it consists of many rows with many columns or one row with one column (i.e., a single attribute value).

In order to retrieve all of the records with a common value in a particular (nonunique) attribute, for example, all of the salespersons with a commission percentage of 10, the command looks the same as when dealing with a unique attribute:

> Select rows from the SALESPERSON relation in which Commission Percentage = 10.

But the result of the operation may include several rows:

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire
137	Baker	10	1995
204	Dickens	10	1998

If the requirement is to retrieve the entire relation, the command would be: Select all rows from the SALESPERSON relation.

The Relational Project Operator

To retrieve what we referred to earlier as a vertical slice of the relation requires the **Project operator.** For example, to retrieve the number and name of each salesperson in the file, the command might look like:

> Project the Salesperson Number and Salesperson Name over the SALESPERSON relation.

The result will be a long, narrow relation:

Salesperson Number	Salesperson Name
137	Baker
186	Adams
204	Dickens
361	Carlyle

If we project a nonunique attribute, then a decision must be made as to whether or not we want duplicates in the result (although, since the result is itself a relation, technically there should not be any duplicate rows). For example, whether:

> Project the Year of Hire over the SALESPERSON relation.

Produces:

Year of Hire
1995
2001
1998
2001

or (eliminating the duplicates in the identical rows) produces:

Year of Hire
1995
2001
1998

would depend on exactly how this hypothetical, informal command language was implemented.

Combination of the Relational Select and Project Operators

More powerful still is the combination of the Select and Project operators. Suppose we apply them serially, with the relation that results from one operation being used as the input to the next operation. For example, to retrieve the numbers and names of the salespersons working on a 10 percent commission, we would issue:

> Select rows from the SALESPERSON relation in which Commission Percentage = 10.
>
> Project the Salesperson Number and Salesperson Name over that result.

The first command "selects out" the rows for salespersons 137 and 204. Then the second command "projects" the salesperson numbers and names from those two rows, resulting in:

Salesperson Number	Salesperson Name
137	Baker
204	Dickens

The following combination illustrates the ability to retrieve a single attribute value. Suppose that there is a need to find the year of hire of salesperson number 204. Since Salesperson Number is a unique attribute, only one row of the relation can possibly be involved. Since the goal is to find one attribute value in that row, the result must be just that: a single attribute value. The command is:

> Select rows from the SALESPERSON relation in which Salesperson Number = 204.
>
> Project the Year of Hire over that result.

The result is the single value:

Year of Hire
1998

Extracting Data Across Multiple Relations: Data Integration

In Chapter 4, the issue of data integration was broached, and the concept was defined. First, the data in the Salesperson and Customer files of Figure 4.5 was shown to be **nonredundant.** Then it was shown that **integrating data** would require the extraction of data from one file and the use of that extracted data as a search argument to find the sought-after data in the other file. For example, recall that finding the name of the salesperson who was responsible for customer number 1525 required finding the salesperson number in customer 1525's record

in the Customer file (i.e., salesperson number 361) and then using that salesperson number as a search argument in the Salesperson file to discover that the sought-after name was Carlyle. The alternative was the combined file of Figure 4.6 that introduced data redundancy.

A fundamental premise of the database approach is that a DBMS must be able to store data nonredundantly while also providing a data integration facility. But it seems that we may have a problem here. Since relations appear to be largely similar in structure to simple, linear files, do the lessons learned from the files of Figure 4.5 and Figure 4.6 lead to the conclusion that it is impossible to simultaneously have nonredundant data storage and data integration with relations in a relational database?

> In fact, one of the elegant features of relational DBMSs is that they automate the cross-relation data extraction process in such a way that it *appears* that the data in the relations is integrated while also remaining nonredundant.

The data integration takes place at the time that a relational query is processed by the relational DBMS for solution. This is a unique feature of relational database and is substantially different from the functional equivalents in the older navigational database systems and in some of the newer object-oriented database systems. In both the older and newer systems, the data integration is much more tightly built into the data structure itself. In relational algebra terms, the integration function is known as the *Join* command.

Now, focus on the SALESPERSON and CUSTOMER relations of Figure 5.14, which outwardly look just like the Salesperson and Customer files of Figure 4.5. Adding the **Join operator** to our hypothetical, informal command style, consider the following commands designed to find the *name* of the salesperson responsible for customer number 1525. Again, this was the query that seemed to be so problematic in Chapter 4.

> Join the SALESPERSON relation and the CUSTOMER relation, using the Salesperson Number of each as the join fields.
>
> Select rows from that result in which Customer Number = 1525.
>
> Project the Salesperson Name over that last result.

Obviously, the first sentence represents the use of the join command. The join operation will take advantage of the common Salesperson Number attribute, which for this purpose is called the *join field*, in both relations. The Salesperson Number attribute is, of course, the primary key of the SALESPERSON relation and is a foreign key in the CUSTOMER relation. Remember that the point of the foreign key is to represent a one-to-many (in this case) relationship between salespersons and customers. Some rows of the SALESPERSON relation *are related* to some rows of the CUSTOMER relation by virtue of their having the same salesperson number. The Salesperson Number attribute serves to identify each salesperson in the SALESPERSON relation, while the Salesperson Number attribute in the CUSTOMER relation indicates which salesperson is responsible for a particular customer. Thus the rows of the two relations that have identical Salesperson Number values are *related*. It is these related rows that the join operation will bring together for the purpose of satisfying the query that was posed.

The join operation tries to find matches between the join field values of the rows in the two relations. For example, it finds a match between the Salesperson Number value of 137 in the first row of the SALESPERSON relation and the Salesperson Number value of 137 in the first, third, fourth, and seventh rows of the CUSTOMER relation. When it finds such a pair of rows, it takes all of the attribute values from both rows and creates a single new row out of them in the resultant relation. In its most basic form, as shown here, the join is truly an exhaustive operation, comparing every row of one relation to every row of the other relation, looking for a match in the join fields. (Comparing every possible combination of two sets, in this case rows from the two relations, is known as the Cartesian product.) So the result of the join command, the first of the three commands in the example command sequence we're executing, is:

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Customer Number	Customer Name	Salesperson Number	HQ City
137	Baker	10	1995	0121	Main St. Hardware	137	New York
137	Baker	10	1995	0933	ABC Home Stores	137	Los Angeles
137	Baker	10	1995	1047	Acme Hardware Store	137	Los Angeles
137	Baker	10	1995	1826	City Hardware	137	New York
186	Adams	15	2001	0839	Jane's Stores	186	Chicago
186	Adams	15	2001	2267	Central Stores	186	New York
204	Dickens	10	1998	2198	Western Hardware	204	New York
361	Carlyle	20	2001	1525	Fred's Tool Stores	361	Atlanta
361	Carlyle	20	2001	1700	XYZ Stores	361	Washington

Notice that the first and seventh columns are identical in all of their values, row by row. They represent the Salesperson Number attributes from the SALESPERSON and CUSTOMER relations, respectively. Remember that two rows from the SALESPERSON and CUSTOMER relations would not be combined together to form a row in the resultant relation unless their two join field values were identical in the first place. This leads to identical values of the two Salesperson Number attributes within each of the rows of the resultant relation. This type of join is called an **equijoin.** If, as seems reasonable, one of the two identical join columns is eliminated in the process, the result is called a **natural join.**

Continuing with the command sequence to eventually find the name of the salesperson responsible for customer number 1525, the next part of the command issued is:

> Select rows from that result (the relation that resulted from the join) in which Customer Number = 1525.

This produces:

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Customer Number	Customer Name	Salesperson Number	HQ City
361	Carlyle	20	2001	1525	Fred's Tool Stores	361	Atlanta

Finally, we issue the third command:
Project the Salesperson Name over that last result.

and get:

Salesperson Name
Carlyle

Notice that the process could have been streamlined considerably if the relational DBMS had more "intelligence" built into it. The query dealt with only a single customer, customer 1525, and there is only one row for each customer in the CUSTOMER relation, since Customer Number is the unique key attribute. Therefore, the query only needed to look at one row in the CUSTOMER relation, the one for customer 1525. Since this row only references one salesperson, salesperson 361, it follows that, in turn, it only needed to look at one row in the SALESPERSON relation, the one for salesperson 1525. This type of performance issue in relational query processing will be covered later in this book in the chapter on physical database design (chapter 8).

EXAMPLE: GOOD READING BOOKSTORES

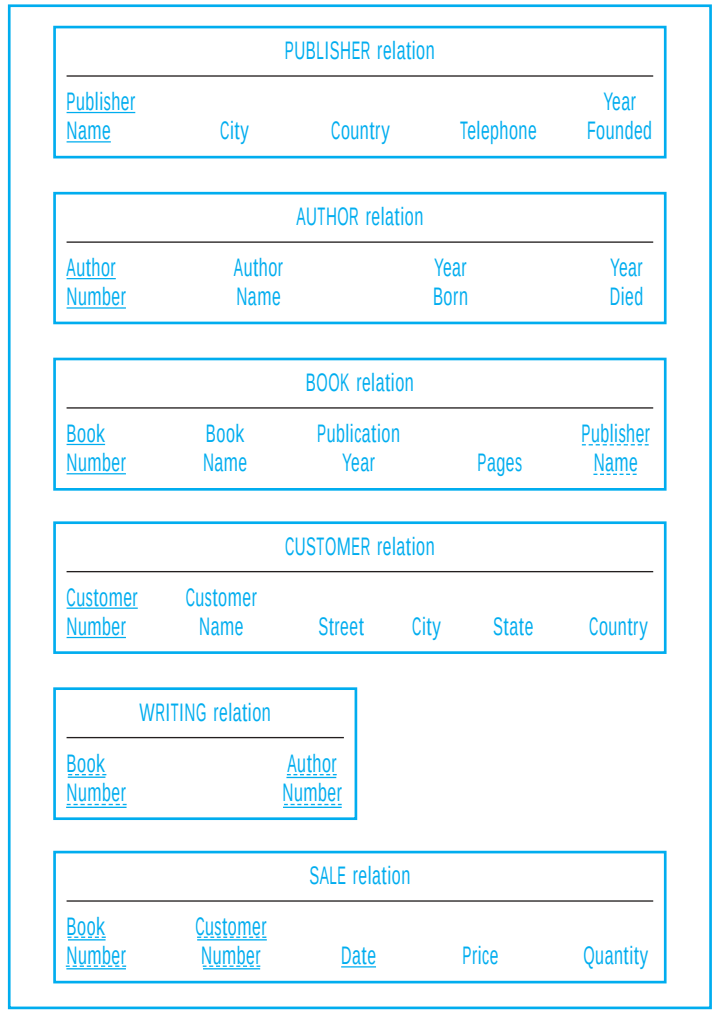
Figure 5.16 shows the relational database for the Good Reading Bookstores example that was described earlier in the book. Since publishers are in a one-to-many relationship to books, the primary key of the PUBLISHER relation, Publisher Name, is inserted into the BOOK relation as a foreign key. There are two many-to-many relationships. One, between books and authors, keeps track of which authors wrote which books. Recall that a book can have multiple authors and a particular author may have written or partly written many books. The other many-to-many relationship, between books and customers, records which customers bought which books.

The WRITING relation handles the many-to-many relationship between books and authors. The primary key is the combination of Book Number and Author Number. There is no intersection data! Could there be a reason for having intersection data in this relation? If, for example, this database belonged to a publisher instead of a bookstore chain, an intersection data attribute might be Royalty Percentage; that is, the percentage of the royalties that a particular author is entitled to for a particular book. The SALE relation takes care of the many-to-many relationship between books and customers. Certainly, Book Number and Customer Number are part of the primary key of the SALE relation, but is the combination of the two the entire primary key? The answer is that it depends on whether the assumption is made that a given customer can or cannot buy copies of a given book on different days. If the assumption is that a customer can only buy copies of a particular book on one single day, then the combination of Book Number and Customer Number is fine as the primary key. If the assumption is that a customer may indeed buy copies of a given book on different days, then the Date attribute must be part of the primary key to achieve uniqueness.

EXAMPLE: WORLD MUSIC ASSOCIATION

Figure 5.17 shows the relational database for the World Music Association example that was described earlier in the book. There is a one-to-many relationship from orchestras to musicians and, in turn, a one-to-many relationship from musicians to degrees. Thus the primary key of the ORCHESTRA relation, Orchestra Name, appears in the MUSICIAN relation as a foreign key. In turn, the primary key of the

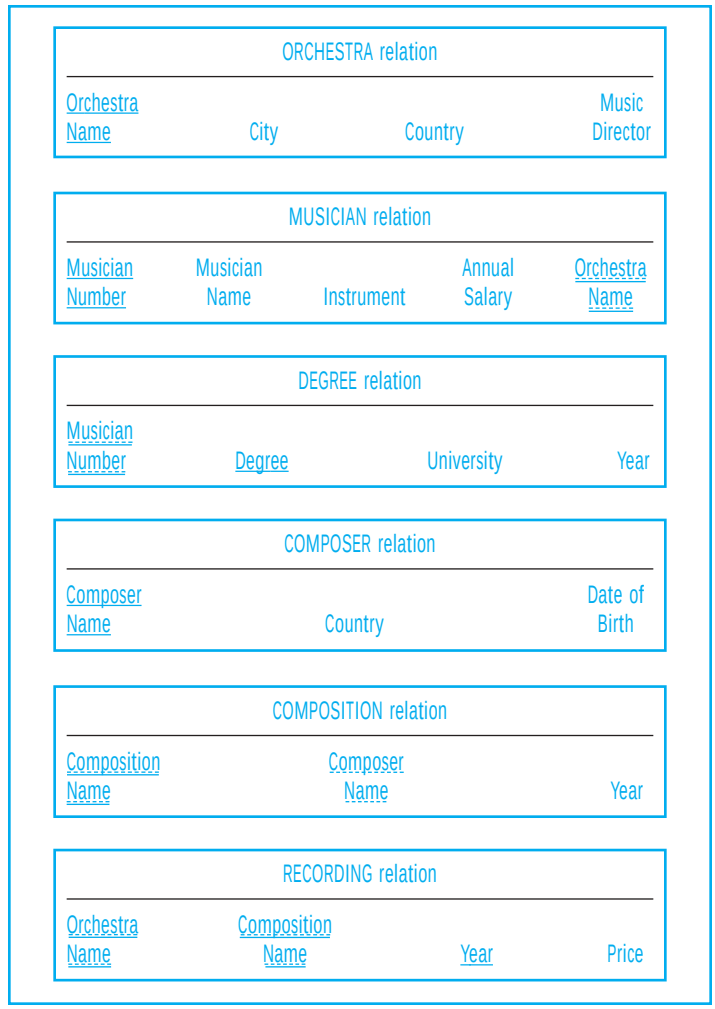
➤ **Figure 5.16**
Good Reading Bookstores relational database



MUSICIAN relation, Musician Number, appears in the DEGREE relation as a foreign key. In fact, since the Degree attribute is only unique within a musician, the Musician Number attribute and the Degree attribute together serve as the compound primary key of the DEGREE relation. The one-to-many relationship from composers to compositions requires that the primary key of the COMPOSER relation, Composer Name, appear as a foreign key in the COMPOSITION relation.

The many-to-many relationship between orchestras and compositions indicates which orchestras have recorded which compositions and which compositions have been recorded by which orchestras. As a many-to-many relationship, it requires that an additional relation be created. The primary key of this new, RECORDING relation has three attributes: Orchestra Name, Composition Name, and Year. Orchestra Name is the unique identifier of orchestras. Composition Name is the unique identifier of compositions. Since a particular orchestra could have recorded a particular composition multiple times in different years (although we assume that this is limited to once per year), Year must also be part of the primary key of the RECORDING relation to provide uniqueness. The Price attribute is then intersection data in the RECORDING relation.

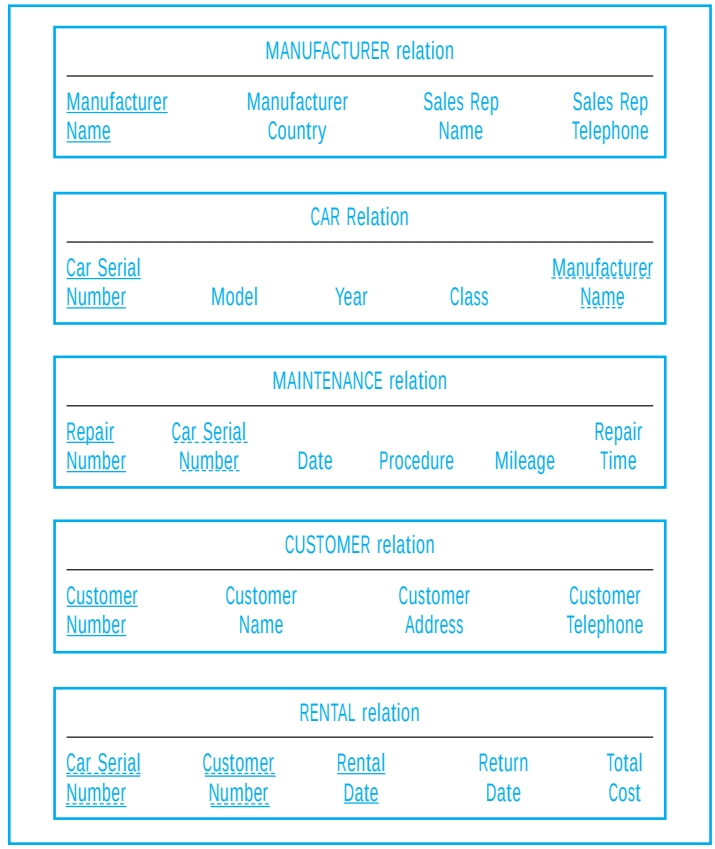
➤ **Figure 5.17**
World Music Association relational database



EXAMPLE: LUCKY RENT-A-CAR

Figure 5.18 shows the relational database for the Lucky Rent-A-Car example that was described earlier in the book. There is a one-to-many relationship from manufacturers to cars and another one-to-many relationship from cars to maintenance events. The former requires the manufacturer primary key, Manufacturer Name, to be placed in the CAR relation as a foreign key. The latter requires the car primary key, Car Serial Number, to be placed in the MAINTENANCE relation as a foreign key. The many-to-many relationship between cars and customers requires the creation of a new relation, the RENTAL relation. Each record of the RENTAL relation records the rental of a particular car by a particular customer. Note that the combination of the Car Serial Number and Customer Number attributes is not sufficient as the primary key of the RENTAL relation. A given customer might have rented a given car more than once. Adding Rental Date to the primary key achieves the needed uniqueness.

➤ **Figure 5.18**
Lucky Rent-A-Car relational database



KEY TERMS

Alternate key
Attribute
Attribute name
Candidate key
Cell
Column
Data retrieval
Domain of values
Equijoin
Foreign key
Integrating data
Intersection data
Join operator
Natural join
Nonredundant data
Personal computer (PC)
Primary key
Project operator
Relation
Relational algebra
Relational database
Row
Select operator
Tuple
Unique attribute

QUESTIONS

1. Why was the commercial introduction of relational database delayed during the 1970s? What factors encouraged its introduction in the early 1980s?
2. How does a relation differ from an ordinary file?
3. Define the terms tuple and attribute.
4. What is a relational database?
5. What are the characteristics of a candidate key?
6. What is a primary key? What is an alternate key?
7. Define the term foreign key.
8. In your own words, describe how foreign keys are used to setup one-to-many binary relationships in relational databases.

9. Describe why an additional relation is needed to represent a many-to-many relationship in a relational database.
10. Describe what intersection data is, what it describes, and why it does not describe a single entity.
11. What is a one-to-one binary relationship?
12. Describe the purpose and capabilities of:
    a. The relational Select operator.
    b. The relational Project operator.
    c. The relational Join operator.
13. Describe how the join operator works.

EXERCISES

1. The main relation of a motor vehicle registration bureau's relational database includes the following attributes:I

Vehicle Identification Number	License Plate Number	Owner Serial Number	Manu-facturer	Model	Year	Color

The Vehicle Identification Number is a unique number assigned to the car when it is manufactured.

The License Plate Number is, in effect, a unique number assigned to the car by the government when it is registered.

The Owner Serial Number is a unique identifier of each owner. Each owner can own more than one vehicle.

The other attributes are not unique.

What is/are the candidate key(s) of this relation? If there is more than one candidate key, choose one as the primary key and indicate which is/are the alternate key(s).

2. A relation consists of attributes A, B, C, D, E, F, G, and H.

No single attribute has unique values.

The combination of attributes A and E is unique.

The combination of attributes B and D is unique.

The combination of attributes B and G is unique.

Select a primary key for this relation and indicate any alternate keys.

3. In the General Hardware Company relational database of Figure 5.14:
    a. How many foreign keys are there in each of the six relations?
    b. List the foreign keys in each of the six relations.
4. Identify the relations that support many-to-many relationships, the primary keys of those relations, and any intersection data in the General Hardware Company database.
5. Consider the General Hardware Company relational database. Using the informal relational command language described in this chapter, write commands to:
    a. List the product name and unit price of all of the products.
    b. List the employee names and titles of all of the employees of customer 2198.
    c. Retrieve the record for office number 1284.
    d. Retrieve the records for the customers headquartered in Los Angeles.
    e. Find the size of office number 1209.
    f. Find the name of the salesperson assigned to office number 1209.
    g. List the product name and quantity sold of each product sold by salesperson 361.
6. Consider the General Hardware Company relational database and the data stored in it, as shown in Figure 5.14. Find the answer to each of the following queries that are written in the informal relational command language described in this chapter.
    a. Select rows from the CUSTOMER EMPLOYEE relation in which Customer Number = 2198.
    b. Select rows from the CUSTOMER EMPLOYEE relation in which Customer Number = 2198. Project Employee Number and Employee Name over that result.
    c. Select rows from the PRODUCT relation in which Product Number = 21765.
    d. Select rows from the PRODUCT relation in which Product Number = 21765. Project Unit Price over that result.
    e. Join the SALESPERSON and CUSTOMER relations using the Salesperson Number attribute of each as the join fields. Select rows from that result in which Salesperson Name = Baker. Project Customer Name over that result.
    f. Join the PRODUCT relation and the SALES relation using the Product Number attribute of each as the join fields. Select rows in which Product Name = Pliers. Project Salesperson Number and Quantity over that result.
7. For each part of Exercise 6, describe in words what the query is trying to accomplish.

MINICASES

1. Consider the following relational database for Happy Cruise Lines. It keeps track of ships, cruises, ports, and passengers. A "cruise" is a particular sailing of a ship on a particular date. For example, the seven-day journey of the ship Pride of Tampa, that leaves on June 13, 2003, is a cruise. Note the following facts about this environment.
   - Both ship number and ship name are unique in the SHIP relation.
   - A ship goes on many cruises over time. A cruise is associated with a single ship.
   - A port is identified by the combination of port name and country.
   - As indicated by the VISIT relation, a cruise includes visits to several ports and a port is typically included in several cruises.
   - Both Passenger Number and Social Security Number are unique in the PASSENGER relation. A particular person has a single Passenger Number that is used for all of the cruises that she takes.
   - The VOYAGE relation indicates that a person can take many cruises, and a cruise, of course, has many passengers.

SHIP relation				
Ship Number	Ship Name	Ship Builder	Launch Date	Gross Wight

CRUISE relation				
Cruise Number	Start Date	End Date	Cruise Director	Ship Number

PORT relation			
Port Name	Country	Number of Docks	Port Manager

VISIT relation				
Cruise Number	Port Name	Country	Arrival Date	Departure Date

PASSENGER relation				
Passenger Number	Passenger Name	Social Security Number	Home Address	Telephone Number

VOYAGE relation			
Passenger Number	Cruise Number	Stateroom Number	Fare

   a. Identify the candidate keys of each relation.
   b. Identify the primary key and any alternate keys of each relation.
   c. How many foreign keys does each relation have?
   d. Identify the foreign keys of each relation.
   e. Indicate any instances in which a foreign key serves as part of the primary key of the relation in which it is a foreign key. Why does each of those relations require a multi-attribute primary key?
   f. Identify the relations that support many-to-many relationships, the primary keys of those relations, and any intersection data.
   g. Using the informal relational command language described in this chapter, write commands to:
      i. Retrieve the record for passenger number 473942.
      ii. Retrieve the record for the port of Nassau in the Bahamas.
      iii. List all of the ships built by General Shipbuilding, Inc.
      iv. List the port name and number of docks of every port in Mexico.
      v. List the name and number of every ship.
      vi. Who was the cruise director on cruise number 38232?
      vii. What was the gross weight of the ship used for cruise number 39482?
      viii. List the home address of every passenger on cruise number 17543.

2. Super Baseball League

   Consider the following relational database for the Super Baseball League. It keeps track of teams in the league, coaches and players on the teams, work experience of the coaches, bats belonging to each team, and which players have played on which teams. Note the following facts about this environment:
   - The database keeps track of the history of all the teams that each player has played on and all the players who have played on each team.
   - The database only keeps track of the current team that a coach works for.
   - Team Number, Team Name, and Player Number are each unique attributes across the league.

- Coach Name is only unique within a team (and we assume that a team cannot have two coaches of the same name).
- Serial Number (for bats) is only unique within a team.
- In the AFFILIATION relation, the Years attribute indicates the number of years that a player played on a team; the Batting Average is for the years that a player played on a team.

TEAM relation

Team Number	Team Name	City	Manager

COACH relation

Team Number	Coach Name	Coach Telephone

WORK EXPERIENCE relation

Team Number	Coach Name	Experience Type	Years of Experience

BATS relation

Team Number	Serial Number	Manufacturer

PLAYER relation

Player Number	Player Name	Age

AFFILIATION relation

Player Number	Team Number	Years	Batting Average

a. Identify the candidate keys of each relation.
b. Identify the primary key and any alternate keys of each relation.
c. How many foreign keys does each relation have?
d. Identify the foreign keys of each relation.
e. Indicate any instances in which a foreign key serves as part of the primary key of the relation in which it is a foreign key. Why does each of those relations require a multi-attribute primary key?
f. Identify the relations that support many-to-many relationships, the primary keys of those relations, and any intersection data.
g. Assume that we add the following STADIUM relation to the Super Baseball League relational database. Each team has one home stadium, which is what is represented in this relation. Assume that a stadium can serve as the home stadium for only one team. Stadium Name is unique across the league.

STADIUM relation

Stadium Name	Year Built	Size	Team Number

What kind of binary relationship exists between the STADIUM relation and the TEAM relation? Could the data from the two relations be combined into one without introducing data redundancy? If so, how?

h. Using the informal relational command language described in this chapter, write commands to:
   i. Retrieve the record for team number 12.
   ii. Retrieve the record for Coach Adams on team number 12.
   iii. List the player number and age of every player.
   iv. List the work experience of every coach.
   v. List the work experience of every coach on team number 25.
   vi. Find the age of player number 42459.
   vii. List the serial numbers and manufacturers of all of the Vultures' (the name of a team) bats.
   viii. Find the number of years of college coaching experience that Coach Taylor of the Vultures has.